Data Management and Domain Adjusting Strategies for Implementing Parallel SPH Method in JPUE Simulation

Zhixuan Cao, Abani Patra
Department of MAE
The University at Buffalo
Buffalo, New York 14260
Email: zhixuanc@buffalo.edu, abani@buffalo.edu

Matthew Jones
Center for Computational Research
The University at Buffalo
Buffalo, New York 14260
Email: jonesm@buffalo.edu

Abstract

This paper presents a parallel implementation of smoothed particle hydrodynamics (SPH) method using the message passing interface (MPI) standard to simulate a turbulent jet or plume which is ejected from a nozzle into a uniform environment (JPUE). Background grid is used to reduce neighbors searching cost and to decompose (redecompose) the domain. Space Filing Curve (SFC) based index is adopted to assign an unique identifier to each background grid. As simulation of JPUE requires adding of new particles during simulation, time-dependent SFC based indexes are assigned to particles to guarantee uniqueness of identifiers. Both particle and background grid are managed by hash tables which ensure quick and flexible accessing, adding and deleting of elements. A SFC based three dimensional domain decomposition and a dynamic load balancing strategy are implemented to ensure good load balance. In addition, computational domain is adjusted during simulation to reduce computational cost. Numerical tests show that our code has acceptable strong scalability and good weak scalability. These strategies can be further applied to many other implementations of meshfree methods, especially those implementations that require irregular particle distributions, adding and deleting of particles and leverage adaptive background grids.

1. Introduction

SPH is a meshless scheme, initially developed for astrophysical applications by Lucy [1] and Gingold and Monaghan [2]. Subsequently it was extended to computational fluid mechanics. A lot of research has appeared on parallellization of SPH in the last few years. Goozee [3] implemented a simple SPH code using MPI, OpenMP and BSP. Wenbo [4] presented a parallel SPH implementation on multi-core CPUs. Holmes [5] presented a simulation framework that enables distributed numerical computing in multi-core shared-memory environments. Dominguez [6] presented optimizations for both CPU and GPU parallelization of a SPH method. Ferrari [7] parallelized a 3D SPH code using the message passing interface (MPI) standard, together with a dynamic load balancing strategy to improve the computational efficiency of the scheme. Kumar et. al. [8] implemented a parallel Godunov SPH in simulation of granular flows. Crespo [9] used the GPUs to accelerate DualSPHysics by up to two orders of magnitude compared to the performance of the serial version.

However, most implementations of parallel SPH method presented to date are limited to standard SPH and benchmark problems like dam break, or relatively simple scenarios like breaking-waves, floodings etc. Work on more complicated problems, such as eruption of volcano plume (Fig. 6) which is essentially a multiphase turbulent, ejection mixing process accompanied by microphysics phenomena like phase change of water, aggregation, reaction etc. is relatively rare. Prediction of such complicated phenomena with acceptable accuracy at given time window requires resolutions (very high particle counts) that cannot been accomplished without parallel computing. What's more, imposing of some types of boundary conditions (such as realistic wind field, eruption boundary condition) requires dynamically adding and removing of particles during simulation. This requires efficient and more flexible data management scheme. What we

need to point out is that the benefit of flexible data management strategies are not only limited to specific implementations of SPH. Actually, flexibility in data management are more critical for several advanced techniques of SPH, such as dynamic particle splitting techniques [10], [11], which will give higher resolution at the area of interest by splitting one large particle to several smaller ones. In this paper, we implement SPH to simulate a simplified version of complicated volcanic plume: the JPUE, and develop data management schemes for it.

Among existing CPU parallel SPH schemes, most of them focus on neighbors searching algorithm and dynamic load balancing. (eg. [7], [9]). Less attention has been paid to developing of more flexible data management schemes for more complicated problems. Fortunately, efficient and flexible data management strategies for high performance computing have been successfully implemented in mesh based methods (eg. [12] for adaptive hp FEM, and [13], [14] for FVM). Motivated by techniques developed for mesh based methods, we present a complete framwork for parallelizing SPH program with MPI standard model allowing more flexible and efficient data access.

Any implementation of SPH code requires efficient searching and updating of neighbors during simulation. Of the many possible choices we adopt a background grid which was proposed by Monaghan and Lattanzio [15] and is quite popular in parallel SPH. The background grid is also used for domain decomposition in SPH. We refer to the elements of background grid, namely squares for two dimension and cubics for three dimension, as buckets. As for the actual storage of data representing the physical quantities associated to each particle, different strategies have been adopted in existing implementations of SPH. In both SPHysics and DualSPHysics [9], the physical quantities of each particle (position, velocity, density...) are stored in arrays, and the particles (and the arrays with particle data) are reordered following the order of the cells. This has two advantages: 1) access pattern is more regular and more efficient, 2) it is easy to identify the particles that belong to a cell by using a range since the first particle of each cell is known. But adding, deleting and especially accessing of particles are not flexible enough. Ferrari [7] adopted linked lists using pointers so that particles can be deleted or added during the simulation. Storage problems caused by fix-size arrays are thereby also eliminated. We define C++ classes which contain all data for particle and bucket. As for the management of data, we adopt hash tables to store pointers to objects of particles and buckets, which give us not only flexibility of deleting and adding element, but also quicker access compared with linked list. Instead of using the "nature manner" to number particles, we adopt SFC based index to give each particle and background bucket an unique identifier – a strategy known to preserve some locality at minimal cost. The SFC based numbering strategy is further extended to include time step information so that particles added at the same position but different time will have different identifiers. As for domain decomposition, even though more complicated graph-based partition tools [16] might get higher quality decomposition, they requires much more effort in programming and computation. So we adopt an easy-programming scheme based on SFC [17].

To the best of the author's knowledge, no implementation of SPH has the feature of adjusting computational domain based on simulation needs. For JPUE simulation, such feature will greatly reduce computational cost by avoiding computing of uninfluenced fluids. This feature is accomplished by adding a scan function to monitor the outermost layer of the domain and turn ghost particles to real particles at proper time.

The data structure, particle and bucket indexing strategies, domain decomposition, dynamic load balancing method and domain adjusting strategies in this paper can be easily adopted by other implementations of any meshfree methods (include SPH). The flexibility of data accessing enables implementing of meshless methods for solving of more complicated problems and using of more advanced techniques.

2. Data Structure and Load Balance

SPH is a meshfree, Lagrangian method. The domain is discretized by particles and the position of each particle is updated at every time step. The physical laws (such as conservation laws of mass, momentum and energy) written in the form of partial differential equations need to be transformed into the Lagrangian particle formalism of SPH. Using a kernel function that provides the weighted estimation of the field variables in the neighborhood of a discrete point (particle), the integral equations are evaluated as sums over neighbor particles. Only particles located within support of kernel function will interact. Thus, physical properties (density, velocity, internal energy...) associated to the particle are updated based on its neighbors. So a neighbor searching needs to be carried out before updating of physical properties. We use buckets which contain all particles associated with a sub-domain and are kept fixed in time during the entire simulation, to reduce searching cost (since search can now be restricted to only neighboring domains). A basic work

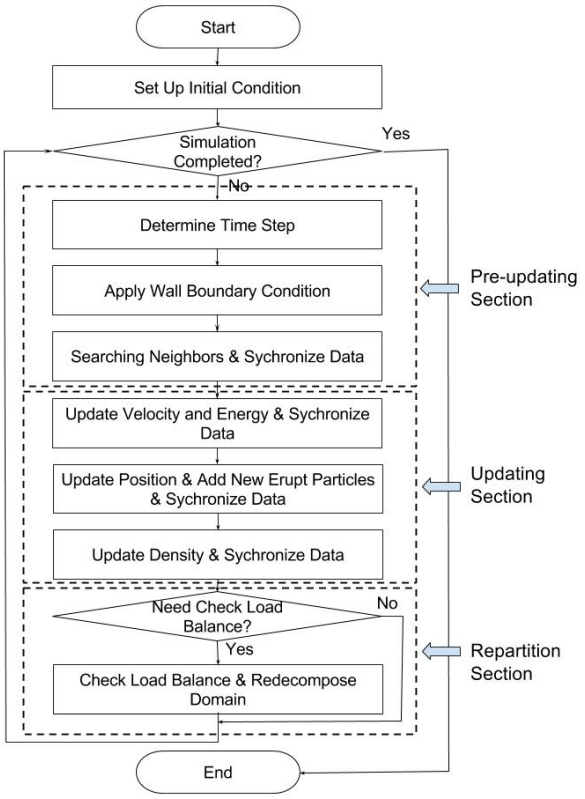


Figure 1. Basic work flow for SPH code

flow of our SPH code is shown in figure 1. Domain is decomposed based on a SFC going through centroids of all buckets. Our data management strategies will also guarantee flexible adding, deleting and accessing of data.

2.1. SFC based indexing

Our data structure starts from assigning each particle and bucket an identifier, we refer to it as key, which should be unique throughout simulation. The key for a bucket is determined by centroid coordinates of the bucket while the key for a particle is determined by adding coordinates and adding time step of the particle. The map from coordinates to key is based on SFC.
The SFC [18] maps a n-dimentional domain to a one dimentional sequence. The standard procedure for obtainning SFC is:

- Scale coordinates into $[0,1]^n$ based on maximum and minimum coordinates of the computational domain: $\mathbf{X}' \to \mathbf{X}$
- Compute $k_r = h_n(\mathbf{X})$. Where h_n is the map $h_n : [0,1]^n \to [0,1]$.
- Convert k_r to integer k by multiplying k_r with a very large number and removing decimal part.
- All keys are sorted to form a sequence which is SFC. The SFC represents a curve passing through all particles (or centroids of buckets).

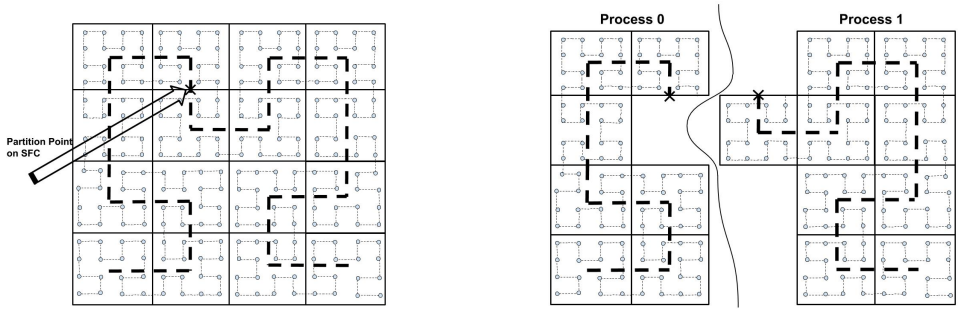
Scheme for constructing the map h_n can be found in [19]. These keys denote a simple addressing/ordering scheme for the data and computations, i.e., a simple global index space for all the objects.
SFC-based indexing scheme can guarantee uniqueness of particle identifier only in simple scenarios when particles are added once while setting up initial condition. In some situations, new particles need to be added while simulation. For example, new particles need to be added at the bottom of the eject vent for JPUE simulation (see figure 3). To distinguish particles added at the "same place" (the small area, all points within which will be mapped to the same k.) at different time steps, we extend the SFC-based key to time-dependent SFC based key by including date of birth of particles into their keys. Then the time-dependent SFC based key can be written as: $[k,t]$, where t is the time step. The map h_n will become:

$$h_n : [0,1]^n \times \mathbf{T} \to [0,1] \times \mathbf{N} \tag{1}$$

Where $\mathbf{T} \subset [0,\infty)$ is the time dimension, $\mathbf{N} = \{0,1,2,3...\}$. To guarantee locality, sorting of particle keys is majorly based on k, that is to say, particle with smaller k always comes before particles with larger k. For these particles have the same k, ordering of them will depend on t. Figure 2.1 shows SFC ordering of buckets and particles in buckets. Several features of such indexing scheme are suitable for SPH:

- Guarantee uniqueness of keys.
- Key of each object is generated purely based on its own coordinates. When add new objects on different processes, key of each object can be generated fast and independently.
- Objects that locate closely in the Euclidian space will also be close to each other in the one dimensional SFC key space in the mean sense. Since SPH particles only interact with its neighbors, geometric locality can be exploited for efficient storage and retrieval of bucket and particle data.
- This type of key effectively generates a global address space. Globality of key and conservation of locality make it easy to partition the sorted key sequence and obtain a decompositiion of the problem.

We need to emphasize here that, in theory, motion of particles will destroy locality established based on initial coordinates of particles. However, as particles



(a) SFCs passing all particles and buckets

(b) Domain decomposition based on SFC of buckets

Figure 2. Spcace filling curve orderings of buckets and particles within the buckets

are moving fairly regularly in a JPUE, the locality of most of particles is largely conserved during simulation. We will revisit this question for more chaotic flows and design a re-numbering method which is computationally cheap, to restore the locality.

2.2. Data structure

2.2.1. Particle and bucket. The most basic data structure of SPH are particle, for problem description, and bucket, for neighbors searches and domain decomposition. Both are defined as classes in C++. Information that is contained in particle class can be categorized into six categories: ID(the key), affiliate(rank of the process that the particle belongs to), primitive variables (variables show up as unknowns in governing equations, e.g. density, velocity), secondary variables (physical properties that can be computed from primitive variables, they are stored to avoid repeated computations eg. pressure, temperature), flags (indicators, such as indicator for ghost particle and real particles, indicator for particles of different phases) and neighbor information (it is a vector of particle keys in our application). Similarly, Information that is contained in a bucket class can also be categorized into different categories: ID, affiliate, dimension information (maximum and minimum coordinates, boundary information), flags (indicators, such as indicator for guest and non-guest, indicator for active and inactive), neighbor information (keys of 27 neighbor buckets including its own key) and owned particles (it is a vector of particle keys in our application). Objects defined based on these two classes are then accessed through hash tables.

2.2.2. Hash table and hash confliction. As discussed at the beginning of this section, implementation of SPH in more realistic scenarios requires dynamic memory management and flexible data access. One of the fundamental data structures that satisfy such requirement is hash table. Another option is B-tree. We adopt hash table in our code. An implementation of B-tree under a similar situation for mesh based methods can be found in other papers (eg. [20]).

Hash table, which is divided into slots, are array based data structure. Based on the key, the address-calculator(hashing) function determines in which slot the data should be stored. The hashing function maps from key to the slot index:

$$slot\,index = hash(key) \tag{2}$$

The hash table has O(1) data access, adding and deleting properties when there is no conflict. How many conflicts will happen depends on both distribution of keys in the key space and size of the hash table. As the distribution of keys is determined by particles' initial locations (or centroids of buckets), the hash table size is under our control. We can use a very large hash table to minimize hash conflicts at the expense of sparse data distribution which will lead to high cache misses and low memory efficiency. Or alternately, we can use smaller hash table size to obtain high memory efficiency on the expense of having more hash conflicts. Patra et al. [20] did numerical experiments to examine the effect of the table size on the different data management operations.

One way to handle hash conflicts is using an additional sorted vector attached to the hash table. When several keys hash to the same slot, a vector will be created. The vector is sorted based on keys so that a binary search can be used to find the correct position for adding, deleting or retrieving. Another option to handle hash conflicts is using an additional link list which is more flexible in memory allocation. The average

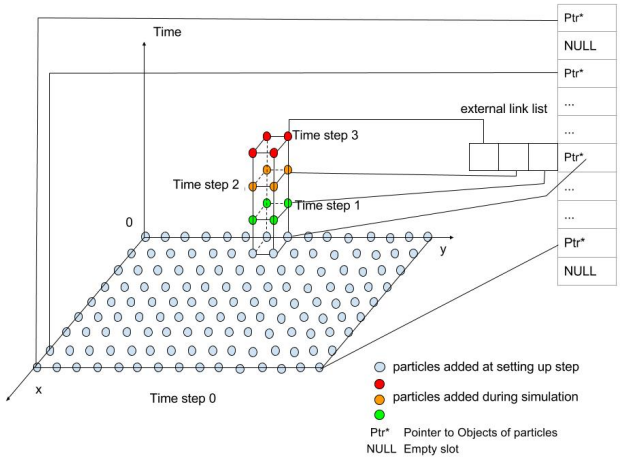


Figure 3. Non-uniform distribution of particles in the $[0,1]^n \times \mathbf{T}$ space due to adding of new particles only at a small portion of the whole domain. These particles are added at different time steps. Pointers to these new particles will be stored in external link lists.

time complexity of binary search is O(log n) while that for linear search based on link list is O(n). However, accessing efficiency of link list is much lower than array based data structure, especially when the link list becomes longer. Choosing of proper way to handle hash conflicts greatly depends on the problem itself. For the test problem in this paper, successively adding particles at the bottom of the eject vent will lead to hash conflicts of many particles, which will lead to a long link list. But considering the conflicts only occur on several slots among millions (see figure 3), the link list is a better choice for us.

2.2.3. Hash function. For time-independent keys, the hash function can be a simple function like:

$$Slot\,Index = \frac{Key - Min\,Key}{Max\,Key - Min\,Key} \times Hash\,Table\,Size \qquad (3)$$

One natural way to hash time-dependent SFC based key $[k,t]$ is to convert the two elements in the key into one number taking k as the higher digit and t as the lower digit of the large number. However, for JPUE simulation, even though ghost particles for wall boundary condition and pressure boundary condition also need to be added during simulation, places for adding of these two types of ghost particles are previsouly empty area (see figure 5). Only ghost particles for eruption boundary condition will be successively

Table 1. Computational Cost Per Particle for Different Steps

Step	Cost (ms)	Abbreviation
neighbor search	0.41	NS
update momentum and energy	0.70	UPME
update density	0.42	UPD
update position	0.01	UPP
velocity filtering	0.43	VF
apply wall bc	0.75	WBC

Table 2. Computational Work Load for Different Types of Particles

Types	NS	UPME	UPD	UPP	VF	WBC	Total
Real	Yes	Yes	Yes	Yes	Yes	No	1.97
wall	No	No	No	No	No	Yes	0.75
eruption	No	No	No	Yes	No	No	0.02
pressure	No	No	No	No	No	No	0.00

added at the same place: bottom of the vent. That is to say, particles are distributed non-uniformly in the $[0,1]^n \times \mathbf{T}$ space as shown in figure 3. To avoid non-uniform, very sparse hash table and conserve locality of SFC, we only plug the first number, k, of the key, $[k,t]$, into the hashing function, equation (3).

2.3. Load balancing strategy

2.3.1. Weighted work load. Particles used in the test problem can be categorized into four types based on the particle-type-flag: real particle, wall ghost particle, pressure ghost particle and eruption ghost particle. Ghost particles are for imposing of corresponding boundary conditions (see figure 5). As different types of particles involve different amount of computational work, shown by table 2 and table 1, we assign different work load weight for different types of particles calibrated by profilling data. Instead of simply using number of contained particles as work load for bucket, work load of each bucket is determined by summing up work load weights of all particles within the bucket. The SFC sequence passing through centroids of all buckets now becomes a weighted sequence. Domain decomposition is conducted based on the weighted SFC of buckets.

2.3.2. Domain decomposition and dynamic load balancing. Figure 2 shows how domain is decomposed based on partition of SFC of buckets. The particles are automatically split into several groups along with buckets that contain them. As SFC of buckets is a curve in the three dimensional space, partition of this curve

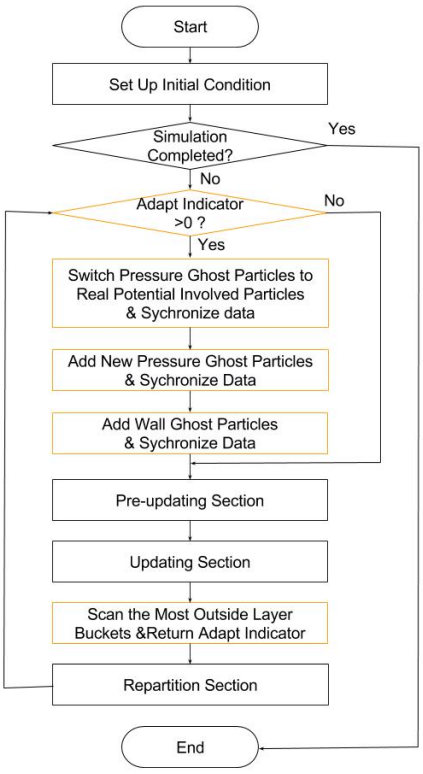


Figure 4. Work flow that enables domain adjusting feature

will automatically lead to 3D domain decomposition. Movement of particles, adding of new particles, adjusting of domain will lead to important load imbalance between processes. To handle this, computational load is monitored at a given interval (The interval is optimized based on numerical experiments). And repartitioning is carried out when load imbalance is larger than a given tolerance.

As some of the neighbor particles reside in other partitions. A set of "guest" particles and buckets are used to synchronize data across partitions. To minimize communications, data is synchronized only where needed, using non-blocking MPI communications.

3. Adjusting of Domain During Simulation

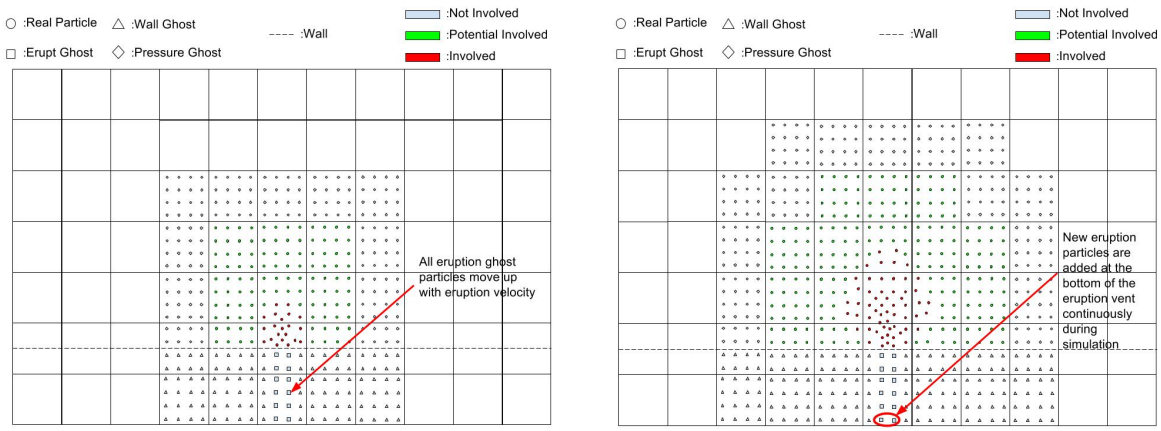
As a Lagrangian method, SPH is able to automatically define and adjust its computational domain as the position of the discretization points are updated at every time step. The collective support of the SPH particles defines the computational domain. However, for JPUE simulation (and many similar/related phenomena), where some fluid ejects into stationary fluid and gets mixed, the domain-adjusting feature of SPH cannot be taken advantage of as all of the stationary fluid must be represented. A lot of CPU time will be spent on computing associated with these stationary particles. If simulating of stationary particles can be avoided, the computational cost will be reduced greatly. We propose a simple strategy to add such feature in our code with low computational cost. We add a scan function to monitor the outermost layer of the domain. When the ejected fluid reaches the boundary of the current domain, ghost particles (for pressure boundary condition) will be turn to real particles and then add new ghost particles for boundary conditions. The original work flow (see figure 1) is modified to enable such a feature (see figure 4).

This is implemented by using a involved-flag, in the particle class. Particles are categorized into three groups based on the value of the involved-flag: involved (involved-flag=2), potentially involved (involved-flag=1), and not involved (involved-flag=0). The involved particles are particles that have already been affected by the mixing. The potentially involved particles are particles that have not been involved in mixing but are adjacent to involved particles and will thus be involved in the near future. All communication and computation associated with uninvolved particles can be ignored. That is to say, only potential involved and involved particles need to be simulated. As simulation progresses, the ejected fluid will reach larger area and more and more particles will be influenced. When originally stationary air is influenced by erupted material, the mass fraction of the erupted material will increase from zero to a positive value. So we can determine whether a particle is involved or not based on whether the mass fraction of that particle is larger than a given threshold (10^{-5} in our simulation). Other physical properties, like velocity, can also serve as alternative "swich criterias".

A similar scheme can also be deployed for buckets resulting in a categorization of buckets. The additional complexity here is that potentially involved ghost particles must also be accounted for. This domain adjusting process is shown in figure 5. The work flow with domain adjusting is in figure 4.

4. Numerical Test

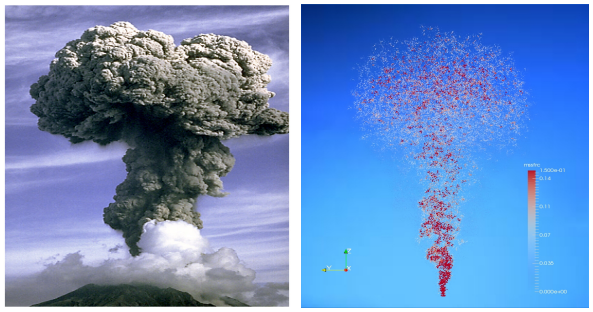
Our target here is to develop data management and parallelization strategies for more complicated implementations of SPH which demand quick and flexible data access, deletion and addition of particles. Thus, the test problem should have such requirements. Simulations of a volcanic eruption, will place these demands on the framework. Thus, we adopt a two phase volcanic



(a) Computational domain at $t = t0$

(b) Computational domain at $t = t0 + \Delta t$

Figure 5. Adjusting the domain based on involved-flag of particles



(a) A plume from Sakurajima Volcano eruption [21]

(b) Simulation results: mass fraction of erupted material at 220 seconds

Figure 6. The volcano plume

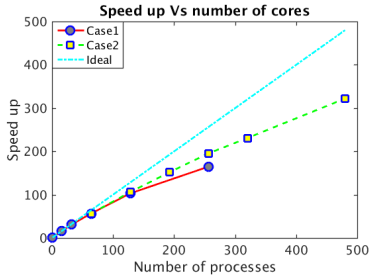


Figure 7. Influence of total work load on strong scalability. It will exceed the time limit on CCR to do a serial simulation for case 2, so the serial simulation time is estimated by assuming that speed up of case 1 and case 2 are the same at 64 processors.

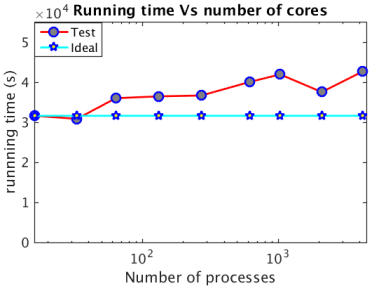


Figure 8. Weak scalability test results

plume model [22] as our test problem. Governing equation, boundary condition and input parameters can be found in [22]. Details about discretization of the governing equations will not be covered in this paper, neither. A typical simulation result of volcano plume by our code is shown in figure 6. Input parameters for this simulation are exactly the same as "Run P1" in reference [22].

4.1. Scalability test

Experiments have been carried out on the computational cluster of Center for Computational Research (CCR) at Buffalo. Three types of processors – 8 core L5520/L5630 running at 2.27/2.13GHz clock rate with 3GB memory per core on a Mellanox Infiniband network and 12 core E5645 running at 2.40GHz clock rate with 4GB memory per core on a Q-Logic Infiniband network. Each node comprise of two sockets with several of these processors. Memory and level 3 cache are shared on each node. The simulational domain is a box with initial ddimension $[-4.8km, 4.8km] \times [-4.8km, 4.8km] \times [0km, 6km]$ for case 1 and $[-10km, 10km] \times [-10km, 10km] \times [0km, 5km]$ for case 2. The smoothing length (we set

Table 3. Simulation Time for Same Particle Weight and Different Particle Weight

Physics time	10 s	20ss	30 s	40 s
Same weight	1141.7	4119.4	10371.0	12453.7
Different weight	1108.2	4057.0	10281.5	12166.3

initial intervals between particles equal to smoothing length) equals to 200m and 100m respectively for test case 1 and test case 2. So the computational work load of test case 2 is larger than that of the test case 1. The simulations run for 20s physical time. Parallel speed up are shown in Fig. 7. Test case2 shows better speed up than test case1 which implies that the overhead of strong scalability can be increased by increasing total amount of work load. The weak scalability test is conducted with the same initial domain as test case 1 and various smoothing length. Each simulation runs for 400 time steps. The average number of real particles of each process keeps constant at 25900, so the average work load for each processor are keep constant. As shown in figure 8, simulation time are almost constant with some minor fluctutations.

4.2. Effect of workload check interval, hash table size and calibrated particle weight

In section 2.3.1, we proposed to use different particle weight for different types of particles when estimate the weight of buckets. The effect of using different particle weight is demonstrated in table 3. The simulation time is reduced by using different particle weights. However, the effect is not as significant as our expectation. One possible explanation is that real particles are the major population, so the load imbalance caused by assignning improper weight value to ghost particles is small.

The best load balance check interval is calibrated based on a series of simulations with different load balance scan/repartition intervals. From 0 - 50 seconds, the interval of 1 second shows a better load balance than interval of 2 second. However, for the simulation of 50 - 100 seconds, interval of 2 second is better. This implies that lose of load balance is faster and requires a more frequent re-decomposition of domain at the initial stage of simulation. This is consistent with the plume development process: the domain grows quickly at beginning as erupted material ejects into the environment and spreads quickly. Afterwards, the spreading speed of the front edge slows down due to viscosity and momentum exchange and domain growth slows down, too.

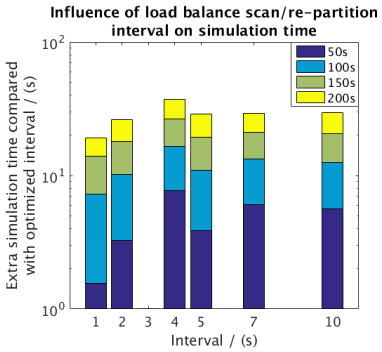


Figure 9. The influence of different load balance check intervals on simulation time. The bar values are the extra simulation time in log scale. The optimized interval is 3s.

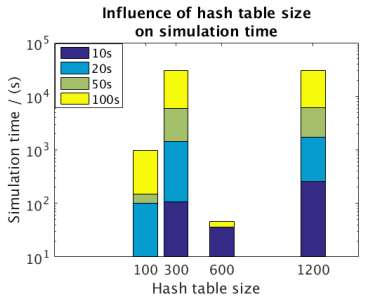


Figure 10. The influence of different bucket hash table size on simulation time. The bar values are calculated by deducting the minimum simulation time among all simulations from simulation time. The y axis is also in log scale.

To figure out the influence of hash table size on performance, a series of simulations up tp 100 seconds of physics time are conducted. Only background buckets hash table size is varied. The results are shown in Fig. 10. Based on these four simulations, the best hash table size should be 600 in a average sense. However, at the first period of simulation, 0 - 10 seconds, hash table size of 100 performs better than 600. Laterly as number of involved buckets increase due to domain expanding, a larger hash table size performs better. We need to emphasize that, these optimized parameters, including weight of particles, load blance checking interval and hashtable size, are case sensitive and need to be re-calibrated for different applications and hardware architecture.

Table 4. Computational Work Load of Extra Steps for Domain Adjusting

Cost	UPME	UPP	ADPP	ADWP	SWCH	SCN
Total time (s)	2954.8	38.55	21.51	8.88	0.08	7.72
Called times	201	201	3	3	2	201

4.3. Domain adjusting

For the test problem in this paper, the volcanic plume will finally reach to a region of $[-10km\ \ 10km] \times [-10km\ \ 10km] \times [0km\ \ 20km]$ after around 300 seconds of eruption. When numerical simulation goes up to 90 seconds, the plume is still within a region of $[-3km\ \ 3km] \times [-3km\ \ 3km] \times [0km\ \ 6km]$. This implies that adjusting of domain can avoid computing large number of uninfluenced air particles, especially for the beginning stage of simulation. On the other hand, additional functions (see Fig. 4) for domain adjusting will add extra computational cost. Table 4 shows the extra computational cost corresponding to these functions. In table 4, SWCH is short for step that switch pressure ghost particle to real particle, ADPP is short for adding new pressure ghost particles, ADWP is short for adding wall ghost particles, SCN is short for scanning the most outside layer of the domain. Profilling data for momentum and energy update (UPME) and position update (UPP) is also included in the table for the purpose of comparison. As we can see from the table, the cost of SCN is even smaller than UPP fucntion, which is the cheapest function (as shown in table 1) in the regular framework (see Fig 1). Other three functions, ADPP, ADWP and SWCH, only called fewer times during the simulation, and as a consequence, the extra computational cost due to them are neglectable. To summarize, the additional cost caused by domain adjusting functions is ignorable. Figure 11 shows that simulation time of the test problem is greatly reduced when we adopt the domain adjusting strategy in our code.

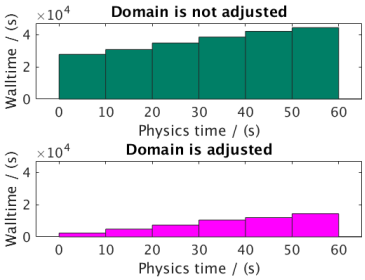


Figure 11. The effect of domain adjusting on simulation time. The figure on the top shows simulation time without domain adjusting, the figure on the bottom shows simulation time with domain adjusting. Different bins represent simulation time up to specific physics time indicated by x axis.

5. Conclusion

We developed data management strategies for parallel implementation of SPH method using MPI standard to simulate complicated problems, such as JUPE, which requires flexible and fast data retrievalling, adding and deleting. Neighbors searching and domain decomposition is based on background grid which overlaps the domain and keep stationary during simulation. SFC based index scheme, which provides a global numberring methodology which is purely coordinates dependent, is adopted to give each bucket an unique identifier. A time dependent key which is also based on SFC is used as identifier for particle. Hash tables with external link list are adopted for accessing particles and buckets data. Based on weighted particle work load, a dynamic load balance strategy is developed by checking load balance and redecomposing the domain at an optimized interval. The performance of the code was further improved to several times faster by adjusting computational domain according to progress of simulation. Scalability tests on our code shew acceptable strong scalability and good weak scalability. It was also demonstrated that 3D domain decomposition provided better strong scalability than 2D domain decomposition when number of processes is larger.

Acknowledgments

Computational results reported here were performed at the Center for Computational Research at the University at Buffalo. This project is supported by Grants No. NSF 1131074 from the National Science Foundation.

References

[1] L. B. Lucy, "A numerical approach to the testing of the fission hypothesis," *The astronomical journal*, vol. 82, pp. 1013–1024, 1977.

[2] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars," *Monthly notices of the royal astronomical society*, vol. 181, no. 3, pp. 375–389, 1977.

[3] R. J. Goozée and P. A. Jacobs, "Distributed and shared memory parallelism with a smoothed particle hydrodynamics code," *ANZIAM Journal*, vol. 44, pp. 202–228, 2003.

[4] C. Wenbo, Y. Yao, and Y. Zhang, "Performance analysis of parallel smoothed particle hydrodynamics on multi-core cpus," in *Cloud Computing and Internet of Things (CCIOT), 2014 International Conference on*. IEEE, 2014, pp. 85–90.

[5] D. W. Holmes, J. R. Williams, and P. Tilke, "A framework for parallel computational physics algorithms on multi-core: Sph in parallel," *Advances in Engineering Software*, vol. 42, no. 11, pp. 999–1008, 2011.

[6] J. M. Domínguez, A. J. Crespo, and M. Gómez-Gesteira, "Optimization strategies for parallel cpu and gpu implementations of a meshfree particle method," *arXiv preprint arXiv:1110.3711*, 2011.

[7] A. Ferrari, M. Dumbser, E. F. Toro, and A. Armanini, "A new 3d parallel sph scheme for free surface flows," *Computers & Fluids*, vol. 38, no. 6, pp. 1203–1217, 2009.

[8] D. Kumar, A. K. Patra, E. B. Pitman, and H. Chi, "Parallel godunov smoothed particle hydrodynamics (sph) with improved treatment of boundary conditions and an application to granular flows," *Computer Physics Communications*, vol. 184, no. 10, pp. 2277–2286, 2013.

[9] A. Crespo, J. Domínguez, B. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, and O. García-Feal, "Dualsphysics: Open-source parallel cfd solver based on smoothed particle hydrodynamics (sph)," *Computer Physics Communications*, vol. 187, pp. 204–216, 2015.

[10] R. Vacondio, B. Rogers, and P. Stansby, "Accurate particle splitting for smoothed particle hydrodynamics in shallow water with shock capturing," *International Journal for Numerical Methods in Fluids*, vol. 69, no. 8, pp. 1377–1410, 2012.

[11] J. Feldman and J. Bonet, "Dynamic refinement and boundary contact forces in sph with applications in fluid flow problems," *International Journal for Numerical Methods in Engineering*, vol. 72, no. 3, pp. 295–324, 2007.

[12] A. Laszloffy, J. Long, and A. K. Patra, "Simple data management, scheduling and solution strategies for managing the irregularities in parallel adaptive hp finite element simulations," *Parallel Computing*, vol. 26, no. 13, pp. 1765–1788, 2000.

[13] E. B. Pitman, C. C. Nichita, A. Patra, A. Bauer, M. Sheridan, and M. Bursik, "Computing granular avalanches and landslides," *Physics of Fluids (1994-present)*, vol. 15, no. 12, pp. 3638–3646, 2003.

[14] A. K. Patra, A. Bauer, C. Nichita, E. B. Pitman, M. Sheridan, M. Bursik, B. Rupp, A. Webber, A. Stinton, L. Namikawa *et al.*, "Parallel adaptive numerical simulation of dry avalanches over natural terrain," *Journal of Volcanology and Geothermal Research*, vol. 139, no. 1, pp. 1–21, 2005.

[15] J. J. Monaghan and J. C. Lattanzio, "A refined particle method for astrophysical problems," *Astronomy and astrophysics*, vol. 149, pp. 135–143, 1985.

[16] R. Biswas and L. Oliker, "Experiments with repartitioning and load balancing adaptive meshes," in *Grid Generation and Adaptive Algorithms*. Springer, 1999, pp. 89–111.

[17] A. Patra and D. Kim, "Efficient mesh partitioning for adaptive hp finite element meshes," in *In International Conference on Domain Decomposition Methods*. Citeseer, 1999.

[18] H. Sagan, *Space-filling curves*. Springer Science & Business Media, 2012.

[19] A. Patra and J. T. Oden, "Problem decomposition for adaptive hp finite element methods," *Computing Systems in Engineering*, vol. 6, no. 2, pp. 97–109, 1995.

[20] A. Patra, A. Laszloffy, and J. Long, "Data structures and load balancing for parallel adaptive hp finite-element methods," *Computers & Mathematics with Applications*, vol. 46, no. 1, pp. 105–123, 2003.

[21] Sakurajima news reports. [Online]. Available: https://www.ucl.ac.uk/vco2/NewsReports/Sakurajima_News_Reports

[22] Y. J. Suzuki, T. Koyaguchi, M. Ogawa, and I. Hachisu, "A numerical study of turbulent mixing in eruption clouds using a three-dimensional fluid dynamics model," *Journal of Geophysical Research: Solid Earth (1978–2012)*, vol. 110, no. B8, 2005.